Procedia Computer Science

Volume 80, 2016, Pages 2281–2286

ICCS 2016. The International Conference on Computational Science

Assessing Run-time Overhead of Securing Kepler

Donghoon Kim[1] and Mladen A. Vouk[2]

[1] North Carolina State University, Raleigh, NC, USA
dkim2@ncsu.edu

[2] North Carolina State University, Raleigh, NC, USA
vouk@ncsu.edu

Abstract

We have developed a model for securing data-flow based application chains. We have implemented the model in the form of an add-on package for the scientific workflow system called Kepler. Our Security Analysis Package (SAP) leverages Kepler's Provenance Recorder (PR). SAP secures data flows from external input-based attacks, from access to unauthorized external sites, and from data integrity issues. It is not a surprise that cost of real-time security is a certain amount of run-time overhead. About half of the overhead appears to come from the use of the Kepler PR and the other half from security function added by SAP.

Keywords: Kepler, security, provenance, data integrity, performance overhead

1 Introduction

Scientific workflow management systems (SWFMS) are in regular use in scientific community. They integrate convenient and powerful components such as graphical user interfaces, distributed IT environments, and automated execution of application and function chains. SWFMS can utilize shared cloud and high performance computing environments in a cost effective way, but if the environment is shared, some concerns may arise.

For example, security can be a major concern when running in a cloud. However, it would appear that security of cloud-based workflows has received a relatively limited attention. On the other hand, many SWFMS have provenance management tools that monitor the workflow and its intermediate results. It is possible to leverage that functionality to implement an approach to securing workflows [1, 2]. The workflow security model we developed does that.

Kepler embraces provenance [3]. We leverage its Provenance Recorder (PR) to capture workflow execution history and through that implement pro-active data-flow security. We have implemented a prototype of what we call Secure Kepler [2] where we modified Kepler provenance module and added to it our Security Analysis Package (SAP) that analyzes provenance information in the security context.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.412

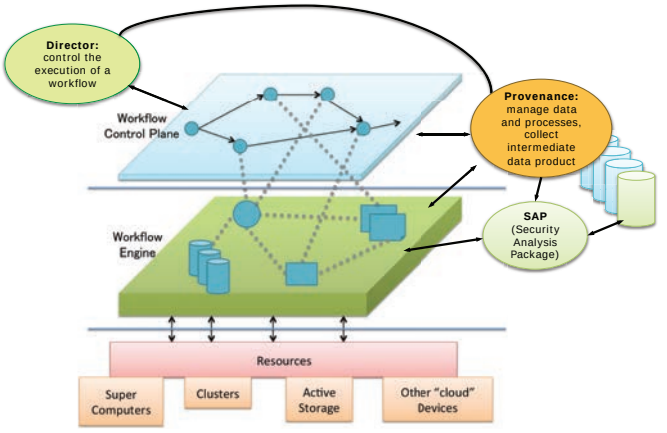


Figure 1: Workflows with SAP

SAP solution was discussed in detail in [2]. In this paper we discuss the overhead that solution may carry. The paper is organized as follows. Section 2 provides background information. Sections 3 and 4 discuss overhead of securing Kepler, and Section 5 concludes this paper.

2 Background

On its own, out-of-the-box Kepler does not have any security modules and it relies on security features of the system where Kepler is executing. We have implemented a prototype of secure Kepler with Security Analysis Package (SAP). SAP is intended to secure input/output flows at the level of the control plane [2]. SAP uses Kepler's provenance module to collect data.

There are three data-flow oriented security property of interest in our model. **Input validation** is used to detect invalid or unauthorized inputs before they are processed by the application (or actor). Some of the most frequent security errors are related to lack of input validation [4]. **Remote access validation** protects Kepler from accessing malicious remote resources. Kepler applications may access external resources and some of those may be compromised. For example, accessing invalid URLs or broken links may lead to invalidated redirects and may invite external attacks [5]. **Data integrity** tests ensure that internal data flows are protected from accessing data stores that may have undergone unintentional of unauthorized changes (e.g., files). When input data and input files are used in read-only, they should be the same as when originally recorded.

In this model, we assume that it is possible to secure the Kepler engine itself (application, its actors, its directors and its database) as well as its provenance module and the information the module collects. However, we allow that an adversary may have access to remote resources that Kepler may use to execute a workflow. In other words, we assume that the top two layers shown in Figure 1 are running in a secure space, and that corruption or attacks if any emanate from the lowest level - external resources. We therefore protect workflow points where data/information flows into a workflow either from some external source, or from an internal data store (e.g., file). We trust the actors that receive inputs only from internal data flows. In complex real-life workflows there may be thousands of actors [6] but only relatively few may be accessing external information and data stores. In that case, while per I/O active actor security analysis overhead may be substantial, the average over the whole workflow may be roughly proportional to the cost of provenance data collections.

SAP "knows" what security properties, if any, should be verified for an actor based on

security verification/validation table added to the provenance module (i.e., database). For example, SAP could use a whitelist to check for acceptable inputs, or to check whether the actor is allowed to access certain external resources. For remote access validation, SAP works as an internal firewall which allows valid URLs and IP addresses that Kepler workflows are allowed to access remotely, and blocks other. For data integrity, SAP conducts a data integrity check before and after Kepler access to stored data. To reduce the overhead of integrity checks, we use a hash value (e.g., MD5). SAP stores the hash value in the provenance security table.

3 Run-time Overhead

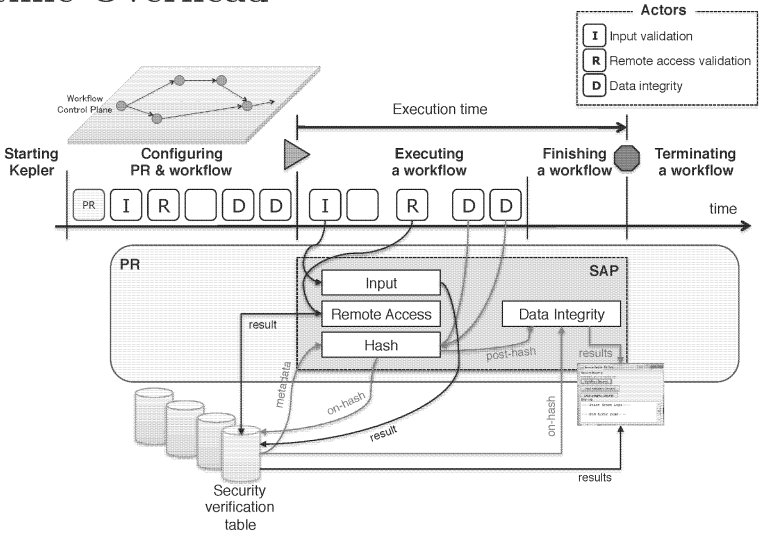


Figure 2: Overhead in Kepler with PR and SAP

So how much does it cost to secure Kepler inputs and data? Provenance Recorder (PR) was designed to keep track of the provenance of data and processes [3]. Performance overhead for PR may vary depending on the amount of provenance data created by a workflow. When a workflow uses PR, a provenance log is automatically created and added to PR during a workflow run. Figure 2 illustrates how SAP works with the operation of checking security properties. We define five steps to explain potential overhead: (1) Staring Kepler; (2) Configuring PR and a workflow SAP probes; (3) Executing a workflow; (4) Finishing a workflow; and (5) Terminating a workflow. SAP is set up with PR in Step 2 (e.g., setting up a security verification table in PR database). PR is executed in Step 2 and 3 while SAP is executed in Step 3 and 4. The actual completion of SAP is the time when a last actor in a workflow which has a security property verified finishes its function. Once PR sends the information from invoked actors to SAP, SAP conducts its function based on an actor's information. **Input validation** and **Remote access validation** are completed in Step 3 and **Data integrity** is completed in Step 4 after comparing on-hash and post-hash values. For example, `I` actor in Figure 2 is an actor scheduled for Input validation. SAP takes an input value, conducts input validation and then sends the validation result to the security verification table. For input validation, SAP compares an input value with values in the whitelist which may lead to a time complexity of $\mathcal{O}(n^2)$, where n is the size of the whitelist. Another example is `D` actor in Figure 2. SAP generates `on-hash` value and sends `on-hash` value and metadata information (e.g., file directory) to the security verification table. On finishing step 3 (`Executing a workflow`), SAP queries metadata

information to the security verification table and generates `post-hash` value. SAP compares `on-hash` from the database with `post-hash` where the overhead is hash computation to generate both `on-hash` and `post-hash`, rather than comparing two hashes ($\mathcal{O}(1)$). If SAP detects a security vulnerability during a workflow execution, SAP can make a workflow execution stop as a way of fault-tolerance that protects Kepler system. With reference to Figure 2 the overhead (O) of SAP can be defined as:

$$O = \sum_{i=1}^{j} I_i + \sum_{i=1}^{k} R_i + \sum_{i=1}^{l} D_i + \delta \tag{1}$$

where (1) I, R, and D is the overhead that occurs when we check on the security properties for input validation, remote access validation, and data integrity, respectively; (2) j, k, and l are the number of actors that require checking of their security properties instead of assuming them. Typically those would be actors that communicate with exterior or with internal or external data stores; and (3) δ is the database access time.

The overhead of input and remote access validation may be smaller than that of checking data integrity because generating hash values may be a costly operation. In addition, SAP needs to generate two hash values (i.e., on-hash and post-hash) to compare before and after values. Of interest is, for example, (a) how much overhead can one expect when SAP is used, and (b) how is this overhead distributed, i.e., how much is the cost of a typical integrity check, a typical input validation, and a typical remote access check.

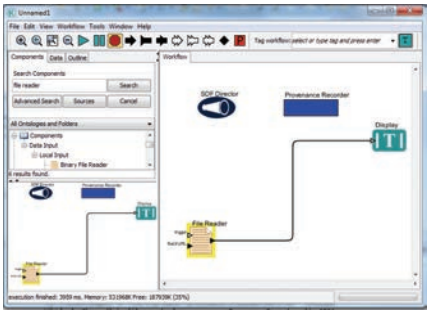
4 An Evaluation

For explanation purposes we use a very simple workflow illustrated in Figure 3. The workflow has one or more `FileReader` actors, and a `Display` actor. The changing element is the amount of data read. We use `FileReader` actor for our micro-benchmark because `FileReader` has two security properties to be verified - input validation and data integrity. First we check if the file has not been corrupted, then we check the inputs to make sure the data read from the file does not cause problems for the receiving applications, e.g., unexpected escape characters. If the file is external to the control plane environment, its integrity would need to be checked every time (even if it is intended to be a read-only file). The same is true of its input impact on the workflow. Kepler application provides the execution time (i.e., `execution finished`) shown in the bottom of the window in Figure 3. We compute and average execution time by running each workflow a number of times.

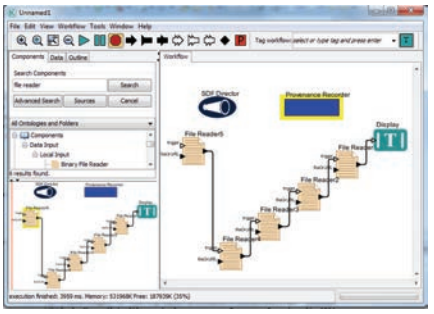
4.1 Experimental Platform

Experiments were conducted in a single desktop computer. Processor is Intel Core Quad CPU 2.40Ghz and RAM is 8GB. We used Kepler 2.4 running on Windows 7 Enterprise (64-bit Operating system) for evaluating the performance overhead of SAP. Figure 3 is the screenshots of a workflow that measures performance overhead. Figure 3a has a `FileReader` actor which is assigned different file sizes from 2MB to 10MB. Figure 3b has five `FileReader` actors, where `FileReader` actors are added one at a time from 1 (one) until 5 (five). The files are generated by a simple program of `Dummy File Creator` [1]. It is worth noting that workflow engine and control plane would typically run in a similar environment (but perhaps on a different platform - e.g.,

[1] http://www.mynikko.com/dummy/



(a) Different file sizes in a `FileReader` actor



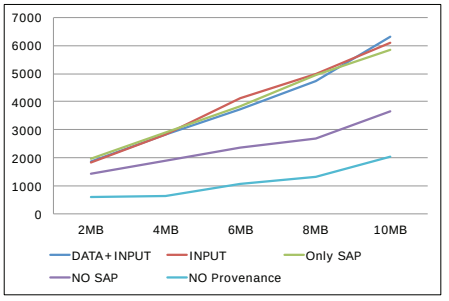
(b) Five `FileReader` actors

Figure 3: A workflow to measure run-time overhead of SAP

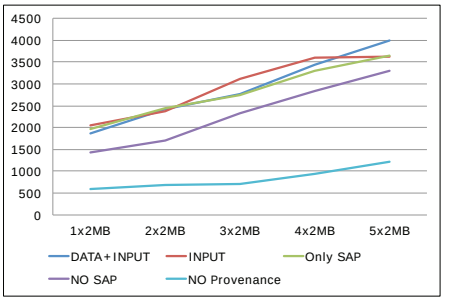
Linux), remote elements of the workflow may be running on clusters, clouds, supercomputers, etc.

4.2 Results

Figure 4 shows average workflow run-time in milliseconds (vertical axis). X-axis in Figure 4a indicates file sizes which the `FileReader` actor should read and X-axis in Figure 4b indicates the number of `FileReader` actors in a workflow, each of which reads a file 2MB in size. For example, "3x2MB" in Figure 4b denotes that a workflow has three `FileReader` actors, each of which reads a file with 2MB in size. Y-axis is the execution time (Time unit: `millisecond`). We note that PR includes PR code execution time and database access time which we call "baseline operating cost". We looked closer at two security properties (i.e., input and data integrity) being checked. We did not consider remote access validation because the implementation of remote access validation is the same as input validation. We conducted the testing with five cases: (1) Input validation (`INPUT`); (2) Data integrity and Input validation (`DATA+INPUT`); (3) only SAP (`Only SAP`); (4) NO SAP - without SAP; and (5) NO Provenance - without PR.



(a) Different file sizes



(b) Different number of `FileReader` actors

Figure 4: Run-time overhead: Input validation and Data Integrity

In Figure 4, we observe that three cases (i.e., `DATA+INPUT`, `INPUT`, and `Only SAP`) are almost the same. Besides visual inspection of the data, we did a quick statistical check to assess how "close" the curves were. There is almost no difference among `DATA+INPUT` checks, `INPUT`

checks and `Only SAP`. We found that core codes for input validation and data integrity checks have very low overhead compared to I/O related overhead. This is a good news in that malformed inputs are often primary sources of security failures [4], Of course, I/O is always more expensive than in-memory operations so it is not surprising that file reading incurs additional overhead. It is somewhat worrying that with PR turned on, Kepler run is perhaps twice as expensive (in this example) as it is without PR, and that if input validation is done rigorously that doubles that overhead. We observe that opening the secure Kepler testing window to show security results incurs overhead. Also, we observe that performance overhead is not confined to operation of checking security properties, but is part of SAP operation - which perhaps is not surprising since SAP draws data from PR even when not acting on any security properties. We are looking into possibly ameliorating the problem. We will be examining more elaborate workflows or real-world workflows to better understand this experimental result.

5 Conclusion

We have examined the level of run-time overhead incurred by our implementation of a security model for Kepler. There are two sources of direct overhead - one is use of the Kepler provenance recorder, the other one is run-time processes that our implementation SAP need. These two costs are about the same. Interestingly, security checks themselves can be a relatively low cost activity. We continue to work on better understanding of the observed run-time overhead, and on finding a less "expensive" solution as well as experimenting on very large workflows to assess scalability properties of SAP model.

5.1 Acknowledgments

This work is supported in part through NSF grants 0910767, 1330553, and 1318564, the U.S. Army Research Office (ARO) grant W911NF-08-1-0105, the NSA Science of Security Lablet, and by the IBM Shared University Research and Fellowships program funding.

References

[1] Daniel Crawl and Ilkay Altintas. A provenance-based fault tolerance mechanism for scientific workflows. In *Provenance and Annotation of Data and Processes*, pages 152–159. Springer, 2008.

[2] Donghoon Kim and Mladen A Vouk. Securing scientific workflows. In *Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on*, pages 95–104. IEEE, 2015.

[3] Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *Provenance and annotation of data*, pages 118–132. Springer, 2006.

[4] MITRE. 2011 cwe/sans top 25 most dangerous software errors, March 2011 (accessed January 2016). http://cwe.mitre.org/top25/.

[5] OWASP. Top 10 2013-top 10, March 2013 (accessed January 2016). https://www.owasp.org/index.php/Top_10_2013-Top_10).

[6] Bertram Ludäscher, Ilkay Altintas, Shawn Bowers, Julian Cummings, Terence Critchlow, Ewa Deelman, David D Roure, Juliana Freire, Carole Goble, Matthew Jones, Scott Klasky, Timorthy McPhillips, Norber Pdohorszky, Claudio Silva, Ian Taylor, and Mladen Vouk. Scientific process automation and workflow management. *Chapter 13 in Scientific Data Management- Challenges, Technology, and Deployment, Editors: Arie Shoshani and Doron Totem*, pages 467–507, 2010, CRC Press.